

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-109788

(P2001-109788A)

(43) 公開日 平成13年4月20日 (2001.4.20)

(51) Int.Cl.<sup>7</sup>

G 0 6 F 17/50

識別記号

F I

G 0 6 F 15/60

テーマコード\* (参考)

6 6 4 K 5 B 0 4 6

6 5 4 A

6 7 2 Z

審査請求 有 請求項の数10 O L (全 18 頁)

(21) 出願番号

特願平11-290276

2/1

(22) 出願日

平成11年10月12日 (1999. 10. 12)

(71) 出願人 000004237

日本電気株式会社

東京都港区芝五丁目7番1号

(72) 発明者 池上 裕之

東京都港区芝五丁目7番1号 日本電気株

式会社内

(74) 代理人 100102864

弁理士 工藤 実 (外1名)

Fターム(参考) 5B046 AA08 BA02 BA03 DA04 GA01

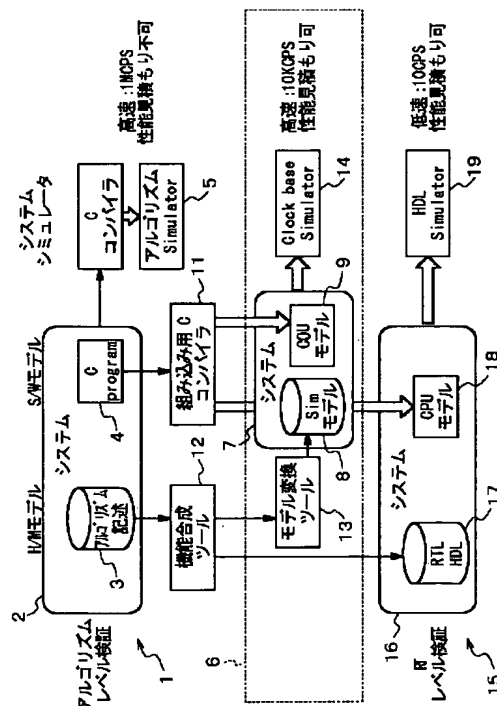
JA04 JA05

(54) 【発明の名称】 シミュレーションモデル、その生成方法、シミュレーション方法及びその記録媒体

(57) 【要約】

【課題】 アルゴリズム記述と R T レベル記述の中間のレベルのシミュレーションを可能にする。その言語の提供。

【解決手段】 資源の制約下、アルゴリズム記述3をクロックレベル記述8に低位化する。アルゴリズム記述3の複数機能を単位クロックの中で動作が可能である部分機能に分解し、その複数機能を回復するためにその部分機能を組み立てる。複数機能は、クロックレベル記述であるクロックレベルシミュレータ8として、レジスタを変数とする言語により表現されている。レジスタを変数とする言語は、クロック単位で動作するプログラミングの言語として最適であり、この言語は、アルゴリズムレベルより下位であり、R T レベルよりも上位の未発見言語であった。



## 【特許請求の範囲】

【請求項 1】 資源の制約の下で、回路のアルゴリズム記述の複数機能を単位クロックの中で動作が可能である部分機能に分解することと、  
前記複数機能を形成するために前記部分機能を組み合わせることにより前記アルゴリズム記述からクロックレベル記述を生成することを含むことを特徴とするシミュレーションモデルの生成方法。

【請求項 2】 前記クロックレベル記述は、  
前記資源のうちのレジスタを異なる複数機能間で兼用させる記述を備える請求項 1 記載のシミュレーションモデルの生成方法。

【請求項 3】 請求項 1 又は 2 に記載されるシミュレーションモデルの生成方法により生成されたクロックレベル記述と、  
前記クロックレベル記述に対応する前記アルゴリズム記述とを対応させ単位クロックに相当する状態毎の資源使用対応表とを含むことを特徴とするシミュレーションモデル。

【請求項 4】 請求項 3 に記載されるシミュレーションモデルを用いて前記単位クロックで前記部分機能を動作させることにより前記複数機能をシミュレートすることを含むシミュレーション方法。

【請求項 5】 前記シミュレートすることの結果に基づいて、状態と前記アルゴリズム記述行とを対応づけ、前記アルゴリズム記述行毎の資源使用状況を表示することを特徴とする請求項 4 記載のシミュレーション方法。

【請求項 6】 前記シミュレーションモデルに含まれるクロックレベル記述と対応表とを用いて、前記アルゴリズム記述行と当該シミュレーションにより得られる変数値の対応を表示することを特徴とする請求項 3 記載のシミュレーション方法。

【請求項 7】 前記クロックレベル記述とクロックレベル記述のうちの前記レジスタに関するレジスタ記述との対応表を作成することと、  
前記クロックレベル記述の前記クロック単位の状態遷移が起これば前記状態遷移が起こった前記アルゴリズム記述部分を表示し、又は、前記クロックレベル記述の前記クロック単位の状態遷移が起これば前記状態遷移が起こった前記レジスタ記述部分の前記レジスタの変数値を表示することの少なくとも一方を実行することを含むことを特徴とする請求項 2 記載のシミュレーションモデルの生成方法。

【請求項 8】 請求項 3 に記載するシミュレーションモデル記述を記録するシミュレーション用記録媒体。

【請求項 9】 アルゴリズム記述と、  
前記アルゴリズム記述よりも抽象度が低い R T レベル記述と、  
前記アルゴリズム記述よりも抽象度が低く、且つ、前記 R T レベル記述よりも抽象度が高いクロックレベル記述

を含み、

前記クロックレベル記述は前記アルゴリズム記述からクロック単位で生成する記述であり、

前記アルゴリズム記述と、前記クロックレベル記述と、前記 R T レベル記述とは、前記アルゴリズム記述に含まれて記述されるレジスタの変数値を共有するシミュレーションモデル。

【請求項 10】 前記クロックレベル記述は演算を含み、時間軸上で進行するクロックの中で実行する前記演算の変数がレジスタである請求項 3 記載のシミュレーションモデル。

## 【発明の詳細な説明】

## 【0001】

【発明の属する技術分野】 本発明は、シミュレーション方法、シミュレーションモデル、これらに対応する記録媒体、それらの記述の生成方法に関し、特に、アルゴリズム記述と R T レベル記述との間でシミュレーションを適宜に切り換えて高速度で所望の段階でシミュレーションモデルを生成してアルゴリズムを高速に又は詳細にシミュレートするシミュレーション方法、シミュレーションモデル、それらに対応する記録媒体、それらの記述の生成方法に関する。

## 【0002】

【従来の技術】 大規模回路は、自動設計装置により設計される。自動設計装置の設計フローは、C 言語のような汎用プログラム言語、専用の動作レベル記述言語により所望の動作フローが記述された最高位レベル記述をレジスタ、加算器のようなハードウェア資源を用いてよりハードウェア化される R T レベル記述のような低位レベル記述に書き下すステップを有している。このようにレベルが異なる抽象度の複数のシミュレーションモデルのシミュレーションは、図 13 (a), (b) に示されるように、抽象度が高い動作レベルシミュレーションモデル 101 では、初めから終わりまでこれだけがシミュレートされ、そのシミュレーションの途中で、R T レベルシミュレーションモデル 102 のシミュレーションは行われえず、また、抽象度が低い R T レベルシミュレーションモデル 102 では、初めから終わりまでこれだけがシミュレートされ、そのシミュレーションの途中で、動作レベルシミュレーションモデル 101 のシミュレーションは行われなかった。

【0003】 このように、動作レベルシミュレーションモデル 101 と R T レベルシミュレーションモデル 102 は、互いに独立していて別個のものとして扱われ、シミュレーションの途中で一方のシステムシミュレーションから他方のシステムシミュレーションに切り換えることは不可能であった。一般的には、速度が重要な局面では高速な上位シミュレーションモデルが用いられ、精度が重要な局面では R T レベル、精度が更に細かく要求される局面ではレベルが更に低い他の低位シミュレーシ

## 3

ンモデルがそれぞれに用いられていた。

【0004】しかし、機能検証のシミュレーションを行う場合、ある時刻  $t_1$  から他のある時刻  $t_2$  の間を特に詳しく調べたいという要望がある。そのような場合にも時刻  $t_0$  から時刻  $t_2$  までの検証のために高精度のシミュレータを使用すると、機能検査に多大な時間がかかってしまう。

【0005】このような問題を解決する技術として、異なる抽象度の複数・シミュレーションモデルの間で切り換えを行うようにしたミックス・シミュレーション技術が、特開平 5-61934 号、論文「並列論理シミュレータ WIZDOM」（情報処理学会第 57 回全国大会、1998）で知られている。ここで開示されている切り換えは、命令レベルシミュレーションモデルとハードウェアシミュレータ、RT レベルシミュレーションモデルのような抽象度が異なるシミュレーションモデルとの間の切り換えである。このような切り換えが可能になっている根拠は、双方のモデルがレジスタ等で共通する構造を持っていることにある。また、ゲートレベルと電子回路レベルの間でのシミュレーション切り換えが行われる技術が、特開平 7-110826 号で知られている。この切り換えで双方の情報のトランスファーは、ゲート回路の端子と電子回路の端子が 1 対 1 に対応していることを利用することにより、回路端子のレベルのアナログ・デジタル変換によって行っている。また、特開平 10-261002 号は、ある回路記述形式を用いてその回路の動きを詳しい定義モデルと粗い定義モデルとで表現し、そのモデルの間での切り換えを開示している。このような 1 対 1 対応の高低レベル間の切り換え、詳細度の異なるものの共通の記述形式を用いた同一レベル間の切り換えが知られている。

【0006】しかし、動作記述レベルといわれる最高位のプログラム言語記述レベルと下位記述レベルである RT レベルの間では、シミュレーション状態を保持する構造が 1 対 1 に対応しておらず、両レベル間でデータの受け渡しができないため、従来、これらのレベル間の切り換えシミュレーションが不可能であった。シミュレーションをより適正に高速化し且つ詳細化するために、このように 1 対 1 に対応する記述がなされていないレベル間のシミュレーションモデルの切替技術の確立が特に望まれた。

【0007】本発明者は、そのような技術の確立のために、論理回路の処理フローを記述した動作レベル記述を機能合成ツールにより RT レベル記述に変換し、その動作レベル記述と RT レベル記述をそれぞれに対応するコンパイラに入力して動作レベルシミュレーションモデル構造と RT レベルシミュレーションモデル構造を生成し、動作レベルシミュレーションモデル構造と RT レベルシミュレーションモデル構造の対応手段を生成し、動作レベルシミュレーションモデル構造と RT レベルシ

## 4

ミュレーションモデル構造に基づいて動作レベルシミュレーションと RT レベルシミュレーションを対応手段を介して切り換えて実行するように、両レベル間でレジスタの変数値を共有させるようにしたシミュレーション方法を提案している（参照：特願平 11-057040 号）。

【0008】このようなシミュレーション方法によるシミュレーションの実行過程で、本発明者は、動作モデルである最高位の抽象度を持つアルゴリズム記述から自動生成される従来の RT レベル記述は、ソースであるアルゴリズム記述から遠くかけ離れ、両記述の間に大きな抽象度の落差が存在していることに気づいた。両記述の間に潜み両記述のそれぞれの抽象度の中間にある抽象度の記述の発見が望まれ、その記述により、アルゴリズム記述より抽象度が低く RT レベルより抽象度が高い言語レベルでシミュレーションモデルを動作レベルより精細に且つ RT レベルより高速にシミュレートすることが望まれる。

## 【0009】

【発明が解決しようとする課題】本発明の課題は、アルゴリズムレベルより抽象度が高く RT レベルよりも抽象度が低いレベルの言語を用いてそれに基づくシミュレーションモデル、その記述とその生成の方法を提供することにある。本発明の他の課題は、アルゴリズムレベルより抽象度が高く RT レベルよりも抽象度が低い記述に基づいて、アルゴリズム記述のシミュレーションよりもより精細に、且つ、RT レベル記述のシミュレーションよりもより高速にシミュレートすることができるシミュレーションモデル、その記述とその生成の方法を提供することにある。

## 【0010】

【課題を解決するための手段】その課題を解決するための手段が、下記のように表現される。その表現中に現れる技術的事項には、括弧（）つきで、番号、記号等が添記されている。その番号、記号等は、本発明の実施の複数・形態又は複数の実施例のうちの少なくとも 1 つの実施の形態又は複数の実施例を構成する技術的事項、特に、その実施の形態又は実施例に対応する図面に表現されている技術的事項に付せられている参照番号、参照記号等に一致している。このような参照番号、参照記号は、請求項記載の技術的事項と実施の形態又は実施例の技術的事項との対応・橋渡しを明確にしている。このような対応・橋渡しは、請求項記載の技術的事項が実施の形態又は実施例の技術的事項に限定されて解釈されることを意味しない。

【0011】本発明によるシミュレーションモデルの記述生成方法は、資源の制約の下で、アルゴリズム記述

(3) をクロックレベル記述 (8) に低位化することを含み、その低位化することは、アルゴリズム記述 (3) の複数機能を単位クロックの中で動作が可能である部分

機能に分解することと、その複数機能を回復するためにその部分機能を組み立てることとを備える。そのような複数機能は、クロックレベル記述であるクロックレベルシミュレータ(8)として、後述されるレジスタを変数とする言語により表現されている。レジスタを変数とする言語は、本発明者により発見された低位プログラミング記述言語である。

【0012】クロックレベル記述は、資源のうちの複数レジスタの内のある1つのレジスタを異なる複数単位クロックのうちで動作させる記述を備える。このようなクロック間でレジスタの共有化が行われ、レジスタが変数として言語化される。

【0013】本発明によるシミュレーションモデルは、更に、既述のこのようなシミュレーションモデルの記述生成方法により記述されたクロックレベル記述(8)と、そのクロックレベル記述に対応するアルゴリズム記述(3)とが対応する対応表(22, 23)とを含む。

【0014】本発明によるシミュレーション方法は、このようなシミュレーションモデルを用いて単位クロックで部分機能を動作させることにより複数機能をシミュレートすることを含み、更に、そのようにシミュレートすることの結果に基づいて、アルゴリズム記述(3)をデバッグすることを含む。更には、シミュレーションモデルに含まれるクロックレベル記述と対応表とを用いて、アルゴリズム記述の行数とシミュレーションにより得られる変数値の対応を表示することを含む。

【0015】本発明によるシミュレーションモデルの生成方法は、資源の制約の下で、アルゴリズム記述(3)をクロックレベル記述(8)に低位化することを含み、その低位化することは、アルゴリズム記述(3)の複数機能を単位クロックの中で動作が可能である部分機能に分解することと、複数機能を回復するためにその部分機能を組み立てることとを備え、クロックレベル記述

(8)は、その資源のうちの複数レジスタの内のある1つのレジスタを異なるクロックのうちでそれぞれに動作させる記述を備え、更に、クロックレベル記述(8)とクロックレベル記述(8)のうちのレジスタ(Reg 1~5)に関するレジスタ記述との対応表(22, 23)を作成することと、クロックレベル記述(8)のクロック単位の状態遷移が起こればその状態遷移が起こったアルゴリズム記述部分を表示することと、クロックレベル記述(8)のクロック単位の状態遷移が起こればその状態遷移が起こったレジスタ記述部分のレジスタ(Reg 1~5)の変数値(R)を表示することを含む。このような対応表の作成によって、両レベル間の移行が可能である。このようなシミュレーションモデルの生成方法に含まれる全記述がコンピュータにより読み取り可能であることは当然であり、それは記録媒体化されて利用に供される。

【0016】本発明によるシミュレーションモデルは、

アルゴリズム記述(3)と、アルゴリズム記述(3)よりも抽象度が低いRTレベル記述(17)と、アルゴリズム記述(3)よりも抽象度が低く、且つ、RTレベル記述(17)よりも抽象度が高いクロックレベル記述

(8)を含み、クロックレベル記述(8)はアルゴリズム記述(3)からクロック単位で自動生成される記述であり、アルゴリズム記述(3)と、クロックレベル記述(8)と、RTレベル記述(17)とは、クロックレベル記述(8)に含まれて記述されるレジスタ(Reg 1~5)の変数値(R)を共有する。このような共有により、3つのレベル間で移行が可能である。

【0017】更に、シミュレーションモデルは、アルゴリズム記述(3)と、アルゴリズム記述(3)よりも抽象度が低いクロックレベル記述(8)を含み、クロックレベル記述(8)は、時間軸に直交する単位クロックごとの断面上で記述されるレジスタの演算記述である。ここで、断面とは、1単位クロック内で同時に動作が進行する一連の記述の連鎖である。アルゴリズム記述よりも抽象度が低く、且つ、時間軸に直交する単位クロックごとの断面上でレジスタに関して記述される言語は、コンピュータにより読み取り可能に表現されて記録媒体化され得る。このような媒体をコピーすることにより、同時並行的に多様なシミュレーションが可能である。アルゴリズム記述よりも抽象度が低く、且つ、時間軸に直交する単位クロックごとの断面上でレジスタに関して記述されるシミュレーションモデル用記述言語は、Reg 1+Reg 2のように複数レジスタが用語として多変数化されている。アルゴリズム記述(3)は数の演算を含んでいる場合には、その演算を時間軸上で進行するクロックの単位の中で実行する関数の変数がレジスタになって表現されている。

【0018】このように、中間レベルの抽象度の言語には、レジスタの動作がクロック単位で現れる変数が潜んでいる。

#### 【0019】

【発明の実施の形態】図1に一致対応して、本発明によるシミュレーションモデルの実施の形態は、アルゴリズム検証部とRTレベル検証部との間で、新たにクロックレベル検証部が設けられていることを特徴としている。そのアルゴリズム検証部1は、図1に示されるように、アルゴリズムシステム2から形成されている。アルゴリズムシステム2は、最も抽象度が高いH/Wモデルであるアルゴリズム記述3と、C言語で記述されS/WモデルであるC-プログラム4とを備えている。アルゴリズムシステム2は、C-コンパイラによりアルゴリズムシミュレータ5に変換される。アルゴリズムシステム2は、アルゴリズムシミュレータ5によりシミュレートされる。

【0020】そのクロックレベル検証部6は、クロックレベルシステム7から形成されている。クロックレベル

システム7は、クロックレベル記述であるクロックレベルシミュレーションモデル8と、クロックレベルCPUモデル9とを備えている。クロックレベルシステム7は、組込用Cコンパイラ11により自動生成されて記述変換され、その記述変換には、慣用の機能合成ツール12が持つツールであるレジスタが用語（単語又は一次変数）として用いられる。アルゴリズム記述3は、機能合成ツール12で記述されるツールを持つモデル変換ツール13によりクロックレベルシミュレーションモデル8に記述変換される。クロックレベルCPUモデル9は、組込用Cコンパイラ11によりCプログラム4から自動生成される。クロックベースシミュレータ14は、クロックレベルシミュレーションモデル8とクロックレベルCPUモデル9とから形成されている。

【0021】そのRTレベル検証部15は、RTレベルシステム16から形成されている。RTレベルシステム16は、RTL-HDL17と、RTレベルCPUモデル18とを備えている。RTL-HDL17は、機能合成ツール12によりアルゴリズム記述3から自動生成される。HDLシミュレータ19は、RTL-HDL17とRTレベルCPUモデル18とから形成されている。

【0022】図2は、クロックレベルシミュレーションモデル8を自動生成するモデル作成のツール構成とその作成フローを示している。機能合成ツール12は、定数、変数の最適化機能と、スケジューリング機能と、アロケーション機能と、レジスタシェアリング機能と、HDL生成機能とを有している。レジスタシェアリング機能は、下記3つの作成を実行する機能を有している：

(1) 資源制約下で用いられる複数レジスタ資源の状態遷移の繰返しをデータパス制御により制御するFSM/Data Pathモデル21の作成

(2) 変数とレジスタと状態位置との対応である変数/レジスタ/状態位置対応表22の作成

(3) アルゴリズム記述のソース行とその状態位置との対応であるソース行/状態位置対応表23の作成

【0023】モデルI/F情報24は、機能合成ツール12とモデル変換ツール13に入力される。モデル変換ツール13は、モデルI/F情報24とFSM/Data Pathモデル21と変数/レジスタ/状態位置対応表22とに基づいて動作し、アルゴリズム記述3をクロックレベルシミュレーションモデル8に記述変換して、クロックレベルシミュレーションモデル8を自動生成する。モデル変換ツール13には、その自動生成のために、モデル機能ライブラリのデバッグ機能ライブラリ25とモデルI/Fライブラリ26からそれらのライブラリ情報が入力される。

【0024】図3は、クロックレベルシミュレーションモデル8の記述構造を示している。クロックレベルシミュレーションモデル8は、Busシミュレーションモデル31を備えている。Busシミュレーションモデル3

1は、モジュールIO部32とデータパス記述部33とから形成されている。モジュールIO部32には、複数IOレジスタ34と複数IOメモリ35とが記述され、モデルI/F情報24から作成され、当該シミュレーションモデルの入出力端子を通して組込ソフトウェアから読み書きされるIOレジスタ構造と、IOメモリ構造を有している。クロックレベルシミュレーションモデル8は、モジュールIO部を介して、他のモジュールと信号伝達を行う。データパス記述部33は、後述されるように、複数IOレジスタと複数演算子との関係を記述するデータパスと、その演算のクロック単位の動作を制御する制御構造とを有している。

【0025】データパス記述部33に、シミュレーションコントローラ36が接続している。シミュレーションコントローラ36は、シミュレータ本体からクロック入力37を受けてデータパス記述部33のクロックを1つつずつ進める演算動作遷移38の循環歩進制御（FSM制御）を行う。シミュレーションコントローラ36から出力されるFSM制御信号39は、データパス記述部33に

【0026】データパス記述部33は、GUIコントローラ41に接続している。データパス記述部33は、レジスタ値Rと演算遷移位置である状態位置値Sとを出力する。レジスタ値Rと状態位置値Sとは、GUIコントローラ41に入力される。リセット信号がシミュレーションコントローラ36に入力されると、データパス記述部33のレジスタ値Rと状態位置値Sとは初期化される。

【0027】GUIコントローラ41は、現在の遷移状態位置のレジスタとアルゴリズムレベル記述による変数との対応を図2に示される変数/レジスタ/状態位置対応表22から得ることができる。GUIコントローラ41は、更に、現在の遷移状態位置のアルゴリズムレベル記述によるソース行をソース行/状態位置対応表23から得ることができる。

【0028】GUIコントローラ41は、アルゴリズムレベル記述変数値表示ウィンドウ42と、アルゴリズムレベル記述ソース実行行表示ウィンドウ43に接続している。アルゴリズムレベル記述変数値表示ウィンドウ42には、レジスタのレジスタ値Rが、その遷移状態位置で、アルゴリズムレベル記述変数値として変数/レジスタ/状態位置対応表22に基づく対応関係が与えられて表示される。複数レジスタが部分的に異なる遷移状態間で共有されることにより一時的にレジスタとの対応が消えてしまうアルゴリズムレベル記述変数の値は、GUIコントローラ41に保持されることにより、アルゴリズムレベル記述変数値表示ウィンドウ42には、常に最新の遷移状態の最新の変数値が表示される。

【0029】他方で、アルゴリズムレベル記述ソース実行行表示ウィンドウ43には、現在の状態遷移位置に

応するアルゴリズムレベル記述のソース行位置値 S が、現在の遷移状態位置で、ソース行／状態位置対応表 23 に基づく対応関係が与えられて、ハイライト表示される。更に、GUI コントローラ 41 は、アルゴリズム記述変数、及び、レジスタ、クロック単位の変化の回数、状態遷移の各状態に遷移した回数、遷移元、遷移先が、シミュレーション中に計測されることにより、変数、レジスタ変化、状態変化の網羅率が測定される。このような網羅率は、テストパターンによりどれ程網羅的な検証が行われ得たかの指標になる。

【0030】アルゴリズム検証：システム内に含まれるハードウェアモデルとソフトウェアモデルは、共にプログラミング言語で記述されている。そのハードウェアモデルは、ハードウェアのアルゴリズムを表現し、そのソフトウェアモデルは組み込みソフトウェアを表現している。これらのプログラミング言語記述を C コンパイラにかけることにより、アルゴリズムレベルのシミュレータが作成される。ハードウェアに関する情報がないアルゴリズムレベルで検証可能な項目は、下記 (1)、(2) に示されるように、純粋に論理的な動作のみであるが、そのシミュレーションは非常に高速である。

- (1) 各モジュールの論理的な動作の検証
- (2) システムの論理的な動作の検証

【0031】クロックレベル検証：ハードウェアモデルのアルゴリズム記述を機能合成ツール、モデル変換ツールで処理することにより、クロックレベルのシミュレーションモデルが作成される。ソフトウェアモデルは、CPU のシミュレーションモデル上に組込ソフトウェアとして読み込まれ、実際の CPU と同等な動作タイミングを実現する。これらのハードウェア、ソフトウェアモデルは、クロックレベルで実際の LSI システムと同等のタイミング精度を持つため、クロックレベルシミュレータ上でシミュレーションを行うことにより、下記のような項目の見積もり、検証が可能である。

- 【0032】(1) 各モジュールのクロックレベルの動作タイミング検証
- (2) 各モジュールのインターフェースの概略検証 (I/O レジスタ、I/O メモリ、I/O 端子の構成、名称、ビット幅等)
- (3) 各モジュール、バスの動作クロックの周波数見積もり
- (4) キャッシュアクセスの見積もり (キャッシュヒット率、アクセス率、ライトバック回数等)
- (5) アクセスの見積もり (バスの占有率、バスのトラザクションごとの Adrs、Data、Master、Slave、Read/Write、Command、語数、占有時間等)
- (6) Bus、Arbiter のアルゴリズムの検証
- (7) メモリ/I/F のトラフィックの見積もり
- (8) データ処理の Through Put の見積もり

- (モジュール毎、バス毎、システム全体)
- (9) バッファ、スタックサイズの見積もり
- (10) 画質、音質の見積もり
- (11) アドレスマップ、端子、bit 幅等モジュール間接続 I/F の整合性の検証
- (12) 浮動小数点を固定小数点に変換した場合の見積もり
- (13) 組み込みソフトウェアの開発、デバッグ
- (14) 消費電力の概略見積もり

10 【0033】RT レベル検証：ハードウェアモデルは機能合成ツールが作成する RTL-HDL モデルが用いられる。このモデルは、クロックレベルよりも詳細に非同期的動作を含むタイミングの精度を持つ。ソフトウェアモデルは、クロックレベルシミュレーションと同様に CPU のシミュレーションモデル上に組み込みソフトウェアとして読み込まれる。これらの RTL-HDL モデルと CPU モデルを HDL シミュレータ上でシミュレートすることにより、以下のような項目の検証と見積もりを行うことができる。

- 20 (1) 各モジュールのインターフェースの詳細検証 (制御、アドレス、データ端子のタイミング動作)
- (2) 各モジュールの詳細タイミングの検証
- (3) 消費電力の詳細見積もり
- (4) テスタ向けパターン作成

#### 【0034】

【実施例】図 10 は、本発明によるシミュレーションモデル作成方法の実施の形態を示している。図 10 に示すフロー図にそって、図 4 に示されるアルゴリズム記述から後述する FSM/Data Path モデル記述及び図 8 に示される変数/レジスタ/状態位置対応表、図 9 に示されるソース行/状態位置対応表を作成し、クロックレベルシミュレーションモデルを作成する方法を以下に述べる。

【0035】図 4 は、アルゴリズム記述 3 (図 2 参照) を例示している。機能合成ツール 12 は、このアルゴリズム記述及び回路を構成する資源の制約条件を入力とする。この場合の資源制約条件は、下記の通りとする。

レジスタ：5 個

加算器：1 個

- 40 図 4 に示されるアルゴリズム記述中に、変数は a、b、c、d、X の 5 個、加算は 3 個が含まれる。資源制約条件から同時に使うことができる加算器は 1 つであるため、演算処理は時分割され、図 5 に示されるようになる。図 5 の Clock 1 では a と b の加算を行い、その結果を X に代入する。Clock 2 では、Clock 1 と同じ加算器を用いて、c と d の加算を行い、その加算結果を一時的に保持するために、変数 t2 を新たに作成して代入する。更に、Clock 3 では Clock 1、2 と同じ加算器を用いて X 及び t2 の加算を行い、その加算結果を X に代入する。このような処理を演算器のス

ケジューリングと呼ぶ（ステップS2）。

【0036】演算器のスケジューリングにより図4のアルゴリズム記述で示される処理を加算器1個の制約下で実現するためには、3クロックを要し、変数はa, b, c, d, X, t2の6個を要することが分かる。レジスタに関する資源制約によれば、同時に利用できるレジスタは5個までとなっているため、レジスタについても共有化を行う必要がある。レジスタ共有化では、まずClock1の変数a, b及びClock2の変数X, c, dにレジスタ制約で許されるReg1からReg5までを割り当てる。Clock3の変数t2については、Reg1からReg5のうちのどれかを使い回す必要がある。Clock1で変数a, bに割り当てられていたReg1, Reg2は、Clock2のXが得られれば、その内容を保持する必要がないため、Clock2以降であれば使い回すことが可能である。ここでは、Clock3の変数t2に割り当てるためReg2を用いることにする。最後の演算結果変数Xについても同様にClock3以降で使い回しができるReg5を割り当てる。このようにして与えられたレジスタ制約の元でレジスタの共有を行う（ステップS3）。図4のアルゴリズム記述にステップS1、ステップS2、ステップS3の処理を行った結果得られるモデルをプログラミング言語で記述したものが下記に示されFSM/DataPathモデルとなる。図5, 6？

【0037】図8は、変数/レジスタ/状態位置対応表22を示している。変数/レジスタ/状態位置対応表22は、図5に示される変数と、Clockの対応、及び、図6に示されるレジスタとクロックの対応から作成される。図5及び図6のClock1からClock3は、図8の状態1～3に対応する。図8の状態1即ち図5、図6のClock1では、レジスタReg1は変数aの値を持ち、レジスタReg2は変数bの値を持つ。状態2では、レジスタReg3は変数X、レジスタReg4はc、レジスタReg5はdの値を持つ。状態3では、レジスタReg2はt2、レジスタReg3はX、状態4では、レジスタReg5はXの値を持つ。更に、初期状態として全てのレジスタに変数の割り当てがない状態0を作成する。このように各状態（即ちClock）における変数とレジスタの対応を示したのが変数/レジスタ/状態位置対応表22である（ステップS5）。

【0038】図9は、ソース行/状態位置対応表23を

```
int DataPath (int a,b,c,d, rest)
{
```

```
    static int FSM position;
    static int Reg1, Reg2, Reg3, Reg4, Reg5;
    if ( rest == 1 ){
        FSM position = 0;
        return (0);
```

示している。Clock1における変数aとbの加算は、図4のアルゴリズム記述では3行目に当たる。Clock2における変数cとdの加算は、図4のアルゴリズム記述では4行目に当たる。Clock3における変数Xとt2の加算も4行目に当たる。最終の演算結果Xが得られる状態は、図4のアルゴリズム記述の5行目に当たる。図8と同様に図9の状態1～3は、図5、図6のClock1～3に対応する。このようにして得られるアルゴリズム記述中の行位置と状態の対応関係を示したのが、ソース行/状態位置対応表23である。モデル変換ツール13にモデルI/F情報24、FSM/DataPathモデル21、変数/レジスタ/状態位置対応表22、ソース行/状態位置対応表23から、クロックレベルシミュレーションモデル8が作成される（ステップS7）。ここで、モデルI/F情報24は、アルゴリズム記述3で表現される回路モジュールと外部とのインターフェースを定義する情報を持つ。その内容としてはバスからアクセス可能なコマンドレジスタや状態レジスタ、データメモリの構成、更には、他のモジュールとの間のデータ転送路のビット幅、同期方式などが含まれる。

【0039】公知慣用のRTレベルシステム16のRTL-HDL17は、図7に示されるように、機能合成ツール12により自動生成される。公知ツールによりこのように自動生成されるハードウェア構成は、5つのレジスタ1～5と、7つのマルチプレクサ1～7と、5つのスイッチSW1～5とから形成される。クロックレベル記述モデルよりも抽象度が低いこのようなハードウェアモデルでは、レジスタの値を保持するためにレジスタの出力をデータ入力にフィードバックさせるフィードバック機能とマルチプレクサとが必要であり、更に、全てのレジスタに常にクロック信号を供給するためのクロック制御と、各マルチプレクサを制御する制御信号44とが必要であり、更に、演算器の共有化のために入力となる複数のレジスタから1つのレジスタを選択するためにもマルチプレクサが必要である。ハードウェアモデルは、後述されるように、その他の各種多様なハード部品とそれを制御する制御信号線が必要である。

【0040】クロックレベルシミュレーションモデル8は、クロック単位で動作する言語として論理レベルの記述言語であるクロックレベル記述が下記ようになされる。

【0041】

```

13      }
      switch (FSM position) {
      case 0:
          Reg1 = a; Reg2 = b; Reg3 = Reg1 + Reg2;
          FSM position = 1;
          break;
      case 1:
          Reg4 = c; Reg5 = d; Reg2 = Reg4 + Reg5;
          FSM position = 2;
          break;
      case 2:
          Reg5 = a; Reg3 = Reg2;
          FSM position = 3;
          break;
      case3:
          X = Reg5;
          FSM position = 0;
          break;

```

14

【0042】Case 1はクロック1のステップに一致し、Case 2はクロック2のステップに一致し、Case 3はクロック3のステップに一致している。このようにクロック単位で、レジスタ1～5が分割されて使用されるクロックレベル記述が生成される。このクロックレベル記述は、図4に示されるアルゴリズム記述による表現よりも表現がより詳細になっているが、後記されるRTレベル記述から見れば、簡素化されている。そのアルゴリズムの内容によるが、アルゴリズム記述のシミュ

entity FF is

```

port(
    Reset:in    std_ulogic;
    Data :in    std_ulogic_vector(31 downto 0);
    Clock:in    std_ulogic;
    Q      :out  std_ulogic_vector(31 downto 0)
);
end FF;

```

【0044】RTハードウェア記述2：

レーション時間はクロックレベル記述のシミュレーション時間に比べて概ね500分の1であり、クロックレベル記述の動作時間はRTレベル記述の動作時間に比べて概ね500分の1であることが本発明者により確認されている。

【0043】図7に対応するVHDL記述が、参考のために下記に記載される。RTハードウェア記述1：

【数1】

【数2】



```

15
architecture A of FF is
begin
  process (Reset, Clock)
  begin
    if (Reset='1') then
      Q <= '0';
    elsif (Clock'event) then
      if (Clock='1') then
        Q <= Data;
      elsif (Clock='X') then
        Q <= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
      endif;
    endif;
  end process;
end A;

```

16

【0045】RTハードウェア記述3:

【数3】

```

entity MUX is
  port (
    A   :in  std_ulogic_vector (31 downto 0);
    B   :in  std_ulogic_vector (31 downto 0);
    Sel :in  std_ulogic;
    Out :out std_ulogic_vector (31 downto 0)
  );
end MUX;

architecture A of MUX is
begin
  process (A, B, Sel)
  begin
    if (Sel='0') then
      Out <= A;
    elsif (Sel='1') then
      Out <= B;
    else
      Out <= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
    endif;
  end process;
end A;

```

【0046】RTハードウェア記述4:

【数4】

17

18

```

entity MUX3 is
  port(
    A  :in  std_ulogic_vector(31 downto 0);
    B  :in  std_ulogic_vector(31 downto 0);
    C  :in  std_ulogic_vector(31 downto 0);
    Sel :in  std_ulogic_vector(1 downto 0);
    Out :out std_ulogic_vector(31 downto 0)
  );
end MUX3;

```

【0047】RTハードウェア記述5:

【数5】

```

architecture A of MUX3 is
begin
  process(A, B, C, Sel)
  begin
    if(Sel='00')then
      Out <= A;
    elsif(Sel='01')then
      Out <= B;
    elsif(sel='10')then
      Out <= C;
    else
      Out <= 'XXXXXXXXXXXXXXXXXXXXXXXXXXXX';
    endif;
  end process;
end A;

```

【0048】RTハードウェア記述6:

【数6】

```

entity ADDER is
  port(
    A  :in  std_ulogic_vector(31 downto 0);
    B  :in  std_ulogic_vector(31 downto 0);
    Out :out std_ulogic_vector(31 downto 0)
  );
end ADDER;

```

【0049】RTハードウェア記述7:

【数7】

19

20

```
architecture A of ADDER is
```

```
begin
```

```
    Out <= A + B;
```

```
end A;
```

```
entity DATAPATH is
```

```
port(
```

```
    a    :in  std_ulogic_vector(31 downto 0);
```

```
    b    :in  std_ulogic_vector(31 downto 0);
```

```
    c    :in  std_ulogic_vector(31 downto 0);
```

```
    d    :in  std_ulogic_vector(31 downto 0);
```

```
    Clock:in  std_ulogic;
```

```
    Reset:in  std_ulogic;
```

```
    X    :out std_ulogic_vector(31 downto 0)
```

```
);
```

```
end DATAPATH;
```

【0050】RTハードウェア記述8: 20 【数8】

```
architecture A of DATAPATH is
```

```
component FF
```

```
port(
```

```
    Reset :in  std_ulogic;
```

```
    Data  :in  std_ulogic_vector(31 downto 0);
```

```
    Clock :in  std_ulogic;
```

```
    Q     :out std_ulogic_vector(31 downto 0)
```

```
);
```

```
end component;
```

【0051】RTハードウェア記述9: 【数9】

```
component MUX
```

```
port(
```

```
    A    :in  std_ulogic_vector(31 downto 0);
```

```
    B    :in  std_ulogic_vector(31 downto 0);
```

```
    Sel  :in  std_ulogic;
```

```
    Out  :out std_ulogic_vector(31 downto 0)
```

```
);
```

```
end component;
```

【0052】RTハードウェア記述10: 【数10】

21

22

component MUX3

port (

A :in std\_ulogic\_vector(31 downto 0);

B :in std\_ulogic\_vector(31 downto 0);

C :in std\_ulogic\_vector(31 downto 0);

Sel :in std\_ulogic\_vector(1 downto 0);

Out :out std\_ulogic\_vector(31 downto 0)

);

end component;

【0053】RTハードウェア記述11:

【数11】

component ADDER

port (

A :in std\_ulogic\_vector(31 downto 0);

B :in std\_ulogic\_vector(31 downto 0);

Out :out std\_ulogic\_vector(31 downto 0)

);

end component;

【0054】RTハードウェア記述12:

【0055】RTハードウェア記述13:

【数12】

【数13】

component CONTROLLER

port (

Clock:in std\_ulogic;

Sel1 :out std\_ulogic;

Sel2 :out std\_ulogic;

Sel3 :out std\_ulogic;

Sel4 :out std\_ulogic;

Sel5 :out std\_ulogic;

Sel6 :out std\_ulogic;

Sel7 :out std\_ulogic;

Sel8 :out std\_ulogic;

Sel9 :out std\_ulogic;

);

end component;

## —Register outputs

```

signal a_out      :std_ulogic_vector(31 downto 0);
signal b_out      :std_ulogic_vector(31 downto 0);
signal c_out      :std_ulogic_vector(31 downto 0);
signal d_out      :std_ulogic_vector(31 downto 0);
signal t1_out     :std_ulogic_vector(31 downto 0);
signal x_out      :std_ulogic_vector(31 downto 0);

```

## —MUX select inputs

```

signal a_muxsel   :std_ulogic;
signal b_muxsel   :std_ulogic_vector(1 downto 0);
signal c_muxsel   :std_ulogic;
signal d_muxsel   :std_ulogic;
signal t1_muxsel  :std_ulogic;
signal add_in1_muxsel :std_ulogic_vector(1 downto 0);
signal add_in2_muxsel :std_ulogic_vector(1 downto 0);
signal x_muxsel   :std_ulogic;

```

## —MUX outputs

```

signal a_muxout   :std_ulogic_vector(31 downto 0);
signal b_muxout   :std_ulogic_vector(31 downto 0);
signal c_muxout   :std_ulogic_vector(31 downto 0);
signal d_muxout   :std_ulogic_vector(31 downto 0);
signal t1_muxout  :std_ulogic_vector(31 downto 0);
signal add_in1_muxout :std_ulogic_vector(31 downto 0);
signal add_in2_muxout :std_ulogic_vector(31 downto 0);
signal x_muxout   :std_ulogic_vector(31 downto 0);

```

## —operator in/out

```

signal add_in1    :std_ulogic_vector(31 downto 0);
signal add_in2    :std_ulogic_vector(31 downto 0);
signal add_out     :std_ulogic_vector(31 downto 0);

```

begin

X <= x\_out;

L\_a\_reg :FF port map(Reset, a\_muxout, Clock, a\_out);

L\_b\_t2\_reg :FF port map(Reset, b\_muxout, Clock, b\_out);

L\_c\_reg :FF port map(Reset, c\_muxout, Clock, c\_out);

L\_d\_reg :FF port map(Reset, d\_muxout, Clock, d\_out);

L\_t1\_reg :FF port map(Reset, t1\_muxout, Clock, c\_out);

L\_x\_reg :FF port map(Reset, x\_muxout, Clock, x\_out);

L\_a\_mux :MUX port map(a, a\_out, a\_muxsel, a\_muxout);

L\_b\_mux :MUX3 port map(b, b\_out, add\_out, b\_muxsel, b\_muxout);

L\_c\_mux :MUX port map(c, c\_out, c\_muxsel, c\_muxout);

L\_d\_mux :MUX port map(d, d\_out, d\_muxsel, d\_muxout);

L\_t1\_mux :MUX port map(add\_out, t1\_out, t1\_muxsel, t1\_muxout);

L\_x\_mux :MUX port map(x, x\_out, x\_muxsel, x\_muxout);

L\_add\_in1\_mux :MUX3 port map(

a\_out, c\_out, t1\_out, add\_in1\_muxsel, add\_in1);

L\_add\_in2\_mux :MUX3 port map(

b\_out, d\_out, t2\_out, add\_in2\_muxsel, add\_in2);

L\_adder :ADDER port map(add\_in1, add\_in2, add\_out);

end A;

【0057】このような記述には、コントローラに関する記述は含まれていない。RTLハードウェア記述14の最初の6行はFFレジスタに関する記述部分であり、その次の6行はマルチプレクサに関する記述部分であり、更にその次の記述はadderによる加算に関する記述部分である。両レベルの記述の両分量は、概ね、両レベルのシミュレーションにかかる時間の長さに対応している。

【0058】図11は、アルゴリズムレベル記述ソース実行行表示ウィンドウ43にアルゴリズム記述ソースを表示する方法を示している。状態遷移が起こる毎に（ステップS11）、GUIコントロール部41は、現在の状態位置をFSM/DataPath部33に問い合わせる（ステップS12）。例えば、現在の状態位置が状態2であった場合、ソース行/状態位置対応表23から、対応するソースファイルがfile1.cで行数が4行目であることが分かる（ステップS13）。このようにして得られたソースファイルをGUI上で表示し、更に現在の状態位置に対応する行をハイライト表示することにより、ユーザーはアルゴリズムレベル記述上での現在の実行位置を示すことができる（ステップS14）。

【0059】図12は、アルゴリズム記述中の変数をアルゴリズムレベル記述変数値表示ウィンドウ42に表示する方法を示している。状態遷移が起こる毎に（ステップS21）、GUIコントロール部41は、現在の状態位置をFSM/DataPath部33に問い合わせる（ステップS22）。例えば、現在の状態位置が状態3であった場合、変数/レジスタ/状態位置対応表22から、状態3で使われる変数はt2とXであり、対応するレジスタがそれぞれReg2とReg3であることが分かる（ステップS23）。GUIコントロール部41は、Reg2とReg3の値をFSM/DataPath部33から得て、開けられている。アルゴリズム記述変数表示ウィンドウ42にはそれぞれt2及びXの値として表示する（ステップS24）。これにより、ユーザーはレジスタ共有が行われていても、GUI上ではクロックレベルシミュレーションモデル中のレジスタ値の変化をアルゴリズムレベル記述上の変数の値の変化として観測することが可能になる。

【0060】クロックレベルシミュレーションモデルは、図7の構造を持つRTL-HDLモデルに比べて、以下のシミュレーションが省略される。

(1) RTL-HDLモデルは、レジスタ値を保持して

それを出力するためのフィードバックとマルチプレクサが必要であるが、プログラミング言語で実現されるクロックレベルシミュレーションモデルでは変数値の保持は代入が起こらない限りにおいて、フィードバックとマルチプレクサが不要である。

【0061】(2)RTL-HDLモデルは、全てのレジスタに常にクロック信号が供給されるためレジスタ値は新しいデータ入力値又はそのレジスタが保持していた値に常に更新されるようになるが、クロックレベルシミュレーションモデルでは新しいデータ入力があった変数のみが更新されるため、不要な更新処理がなくシミュレーションが高速化され得る。

【0062】(3)RTL-HDLモデルは、演算器の共有が行われるため、入力となる複数のレジスタから1つのレジスタを選択するマルチプレクサが必要であるが、クロックレベルシミュレーションモデルでは演算器に制限がなくそのような共有化が不要であり、演算器の入力を選択するマルチプレクサが不要である。

(4)クロックレベルシミュレーションモデルでは演算器共有化のためのマルチプレクサが不要であるので、マルチプレクサに必要であった制御入力を作成する回路が不要である。

【0063】(5)RTL-HDLモデルは、全レジスタに非同期リセット信号がつくが、クロックレベルシミュレーションモデルではクロック周期単位の動作のみを扱うので、非同期的な動作を行う必要がない。

【0064】(6)RTL-HDLモデルは、レジスタ、マルチプレクサ、演算器等のサブモジュールを利用した構造を持ち、そのサブモジュールは端子を持ち、その内部には信号線、制御機構等も持っているが、クロックレベルシミュレーションモデルでは、レジスタはプログラミング言語の変数に、マルチプレクサは条件文に、演算器は演算子で表現されるため、シミュレーション処理は大幅に簡略化される。

【0065】(7)RTL-HDLモデルは、実際のハードウェア通りに束線のビット位置の昇降順、符号のありなし、整数型/ビットベクタ型等の信号型の区別、又は、これらの間の変換を厳密に表現する必要があるが、このような区別は、クロックレベルシミュレーションモデルでは厳密に行われない。

【0066】(8)RTL-HDLモデル内にある各サブモジュール間の動作の並列性は厳密に表現されるが、クロックレベルシミュレーションモデルでは各サブモジュールの厳密な動作タイミングの差に基づいた検証は行われないため、並列性を厳密に表現する必要がない。

【0067】(9)RTL-HDLモデルは、クロックの変化タイミング以外にも例えばリセット信号が変化した時点の動作を厳密に表現するが、クロックレベルシミュレーションモデルではクロック単位の動作のみを扱うので、非同期的な動作に対する処理を簡略化することが

できる。

#### 【0068】

【発明の効果】本発明によるシミュレーションモデル、その記述とその生成の方法は、アルゴリズムレベルより抽象度が低くRTレベルよりも抽象度が高いレベルの言語の発見により、中間レベルのシミュレーションが可能になる。アルゴリズムレベルより抽象度が低くRTレベルよりも抽象度が高い記述に基づいて、アルゴリズム記述のシミュレーションよりもより精細に、且つ、RTレベル記述のシミュレーションよりもより高速にシミュレートすることができる。一般的には、より物理的でありより詳細に記述されるハード部品のシミュレーション言語を論理化してハード部品を変数とするプログラミング言語を作成することが可能であり、公知の複数レベルの記述の中間言語を作成することができる。

#### 【図面の簡単な説明】

【図1】図1は、本発明による論理シミュレーションシステムの実施の形態を示すシステム図である。

【図2】図2は、本発明によるシミュレーションモデル生成方法の実施の形態を示すブロックフロー図である。

【図3】図3は、2レベル間の移行のための対応を示す論理ブロック図である。

【図4】図4は、アルゴリズム記述の実施例を示す論理式である。

【図5】図5は、クロックレベルのスケジューリングをしめす動作時間表である。

【図6】図6は、クロックレベルのレジスタの動作を示す動作時間表である。

【図7】図7は、公知のハードウェア記述を示す回路図である。

【図8】図8は、状態遷移位置数とレジスタ変数値の対応を示すテーブルである。

【図9】図9は、状態遷移位置数とソース行との対応を示すテーブルである。

【図10】図10は、本発明による論理シミュレーション方法の動作の実施の形態を示すフローチャートである。

【図11】図11は、本発明による論理シミュレーション方法の他の動作を示すフローチャートである。

【図12】図12は、本発明による論理シミュレーション方法の更に他の動作を示すフローチャートである。

【図13】図13は、公知の論理シミュレーション方法を示すシステムフロー図である。

#### 【符号の説明】

3…アルゴリズム記述

8…クロックレベル記述（クロックレベルシミュレーションモデル）

22、23…対応表

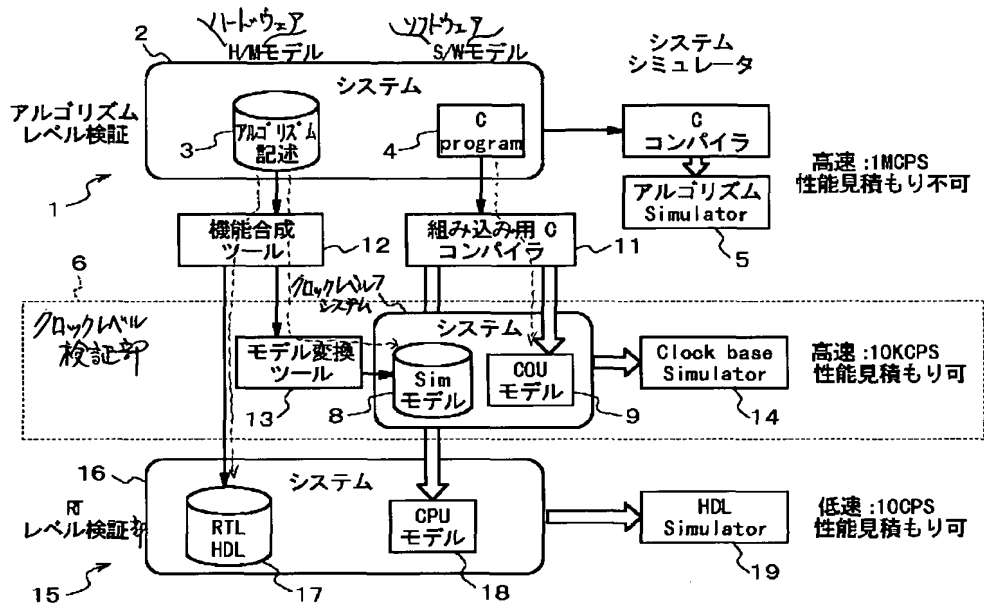
22…変数/レジスタ/状態位置対応表

23…ソース行/状態位置対応表

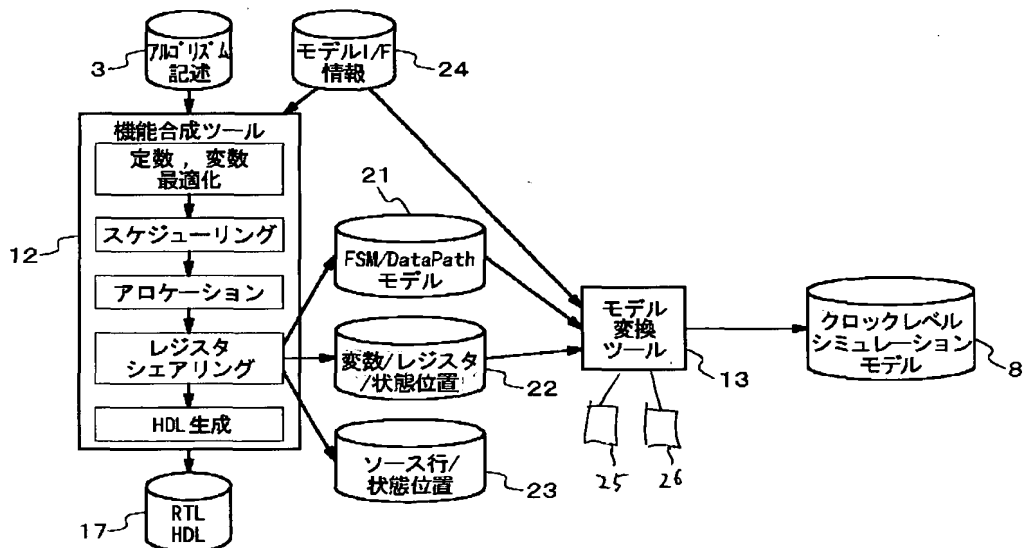
17...RTLレベル記述  
R...変数値

Reg1~5...レジスタ

【図1】



【図2】



【図4】

アルゴリズム記述

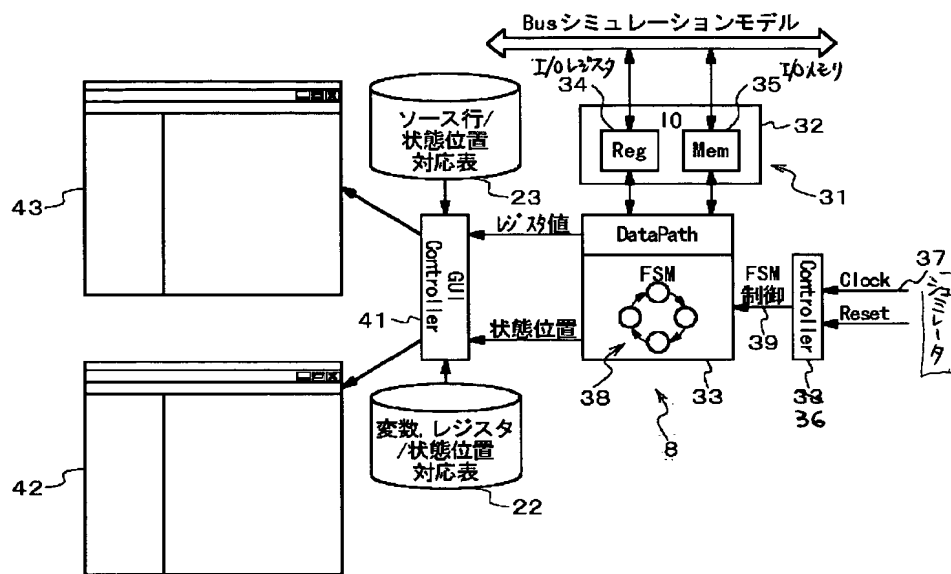
```

1: int DataPath(int a, b, c, d)
2: {
3:   X = (a+b);
4:   X = (c+d) + X;
5:   return(X);
6: }

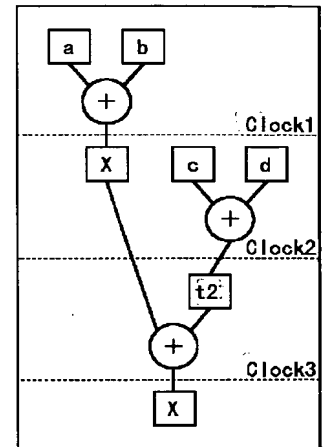
```



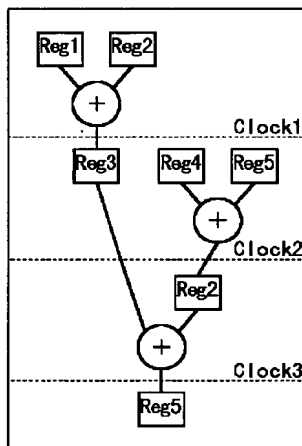
【図 3】



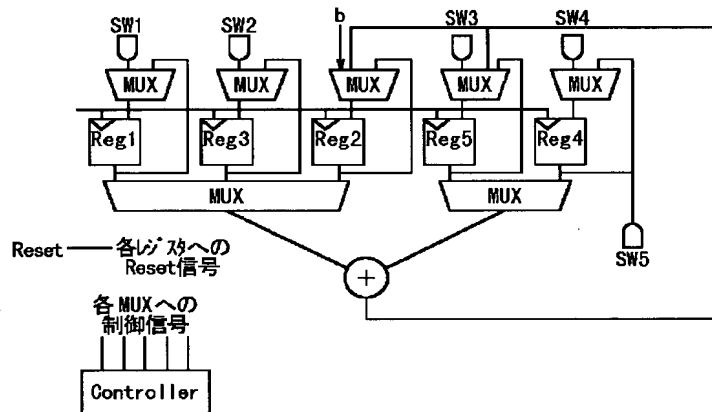
【図 5】



【図 6】



【図 7】



【図 8】

対応表 22

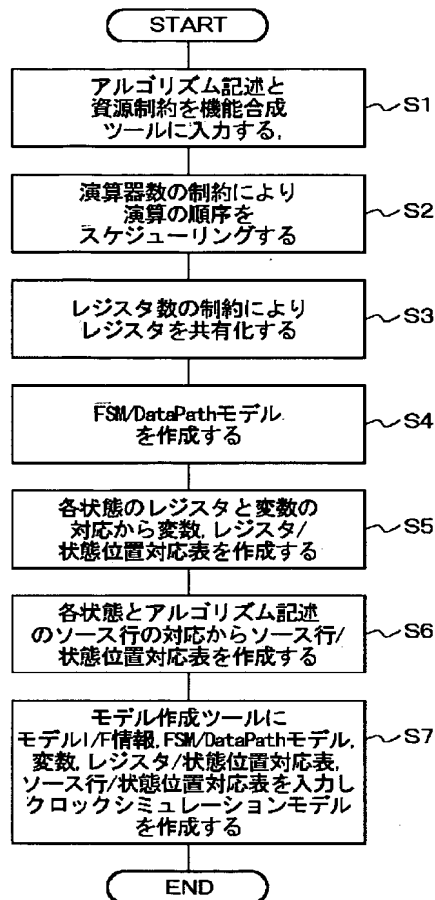
	Reg1	Reg2	Reg3	Reg4	Reg5
状態 0	—	—	—	—	—
状態 1	a	b	—	—	—
状態 2	—	—	X	c	d
状態 3	—	t2	X	—	—
状態 4	—	—	—	—	X

【図 9】

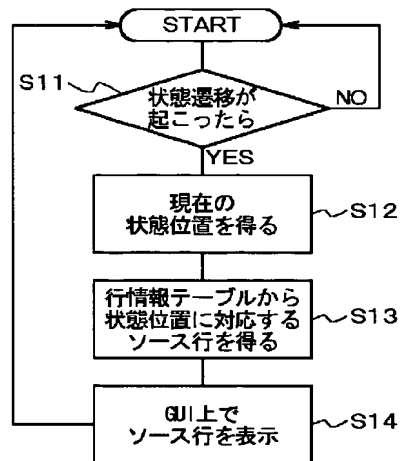
対応表 23

	対応ソース行
状態 0	—
状態 1	File1. c : 3
状態 2	File2. c : 4
状態 3	File3. c : 4
状態 4	File4. c : 4

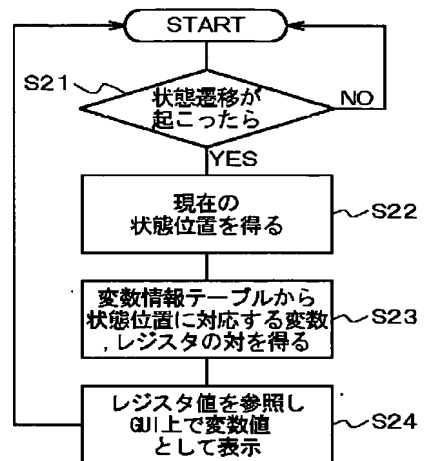
【図10】



【図11】



【図12】



【図13】

